User manual for MINLP_BB *

Sven Leyffer[†]
Argonne National Laboratory

April 1998
Revised, March 1999 and July 2003

Abstract

A software package for the solution of Mixed Integer Nonlinear Programming (MINLP) problems is described. The package implements a branch-and-bound solver with depth-first search and maximal fractional branching.

Key words: Mixed Integer Nonlinear Programming, branch-and-bound.

1 Introduction

The software package `MINLP_BB` described in this note solves MINLP problems by branch-and-bound. These are Nonlinear Programming (NLP) problems in which some of the variables are restricted to take integer values. The nonlinear part of the problem is specified in the same way as for the NLP solver filterSQP [2].

The solver guarantees to find global solutions, if the problem is convex. `MINLP_BB` is also effective to solve non-convex MINLP problems. Even though no guarantee can be given that a global solution is found in this case, the solver is more robust than outer approximation or Benders Decomposition which usually cut away large parts of the feasible region.

`MINLP_BB` can also be used to solve problems with discrete variables (e.g. $z \in \{0.2, 7.4, 18.7\}$). In this case the problem can be reformulated by replacing z by $z = 0.2\,y_1 \;+\; 7.4\,y_2 \;+\; 18.7\,y_3$ and $y_1 + y_2 + y_3 = 1$ where $y_i \in \{0, 1\}$. This is in fact an example of a Special Ordered Set of type 1 (SOS1), e.g. [3].

---

*This work was supported by EPSRC grant number GR/K51204.

[†] `leyffer@mcs.anl.gov`

2 The Algorithm

The package implements a branch-and-bound scheme (e.g. [1]) using a depth-first-search. The resulting NLP relaxations are solved using filterSQP. The user can influence the *branching decision* by supplying priorities for the integer variables. By default, the solver branches on the variable with the highest priority first. If there is a tie, then the variable with the largest fractional part is selected for branching.

3 System Requirements and Implementation

The software package requires a `FORTRAN 77` compiler. It comprises a suite of MINLP subroutines:

`MINLPdriver.f`	A sample driver for the MINLP solver.
`minlpbb.f`	The main MINLP_BB routine.
`minlpbbaux.f`	Auxiliary routines used in `minlpbb.f`.
`BBaux.f`	Auxiliary routines used for MINLP and MIQP.
`MINLPuser.f`	The user supplied problem functions.

In addition the user requires an NLP solver (filterSQP) consisting of:

`filter.f`	The main SQP filter routine.
`filteraux.f`	Auxiliary routines used in `filter.f`.
`QPsolved.f`	The interface to the QP solver, dense storage.
`QPsolves.f`	The interface to the QP solver, sparse storage.
`scaling.f`	Routines that scale the problem.
`bqpd.f`	The main QP solver routine.
`auxil.f`	Some auxiliary routines for `bqpd`.
`denseL.f`	Dense linear algebra package.
`sparseL.f`	Sparse linear algebra package.
`util.f`	Some linear algebra utilities.
`sparseA.f`	Sparse matrix storage/handling **OR**
`denseA.f`	Dense matrix storage/handling.

A `makefile` for `UNIX` systems is supplied with the distribution version. This `makefile` compiles and links the small MINLP problem in [2]. Interfaces to `CUTE` and `AMPL` can be made available upon request.

4 Description of the Interface

The interface of the MINLP solver has the following form. Here `REAL` is Fortran `double precision` by default but can be changed to standard single precision using teh supplied tools.

```
      subroutine minlpsolver(nivar,n,m,kmax,nstackmax,mlp,bl,bu,fstar,
     .                      rho,x,s,lam,ivar,priority,nSOS1,tSOS1,pSOS1,
     .                      iSOS1,rSOS1,SOS1priority,c,cstype,a,la,maxa,
     .                      iwork,liwork,work,lwork,user,iuser,iter,
     .                      iprint,nout,ifail,max_NLP)
```

4.1 Definition of Parameters

A detailed description of the parameters follows below (the parameters preceded by a * must be set on entry to `minlpsolver`.

*	`nivar`	number of integer variables (`INTEGER`)
*	`n`	total number of variables (`INTEGER`)
*	`m`	number of constraints (linear and nonlinear, excluding simple bounds) (`INTEGER`)
*	`kmax`	maximum size of null-space ($\leq$ `n`) (`INTEGER`)
*	`nstackmax`	maximum size of the stack, storing information during the tree-search (`INTEGER`)
*	`mlp`	maximum level of degeneracy in QP solver (`INTEGER`)
*	`bl`	`bl(n+m)` vector of lower bounds (`REAL`)
*	`bu`	`bu(n+m)` vector of upper bounds (`REAL`)
	`fstar`	optimum objective function value (`REAL`)
*	`rho`	initial trust-region radius (`REAL`)
	`x`	`x(n)` optimal integer feasible solution (i.f.s.); or if (ifail=6) the first i.f.s. obtained (`REAL`)
	`s`	`s(n+m)` scale factors for variable/constraint scaling (`REAL`)
	`lam`	`lam(n+m)` Lagrange multipliers of simple bounds and general constraints at solution (`REAL`)
*	`ivar`	`ivar(nivar)` vector of indices of the integer variables (`INTEGER`)
*	`priority`	`priority(n)` is the priority of the integer variables; `priority(ivar(i))` is the priority of variable `x(ivar(i))`; a higher value implies a higher priority (`INTEGER`)
*	`nSOS1`	number of variables that are elements of a SOS1 set (`INTEGER`)
*	`tSOS1`	number of SOS1 sets (`INTEGER`)
*	`pSOS1`	`pSOS1(tSOS1+1)` are pointers to start of each SOS1 (`INTEGER`)
*	`iSOS1`	`iSOS1(nSOS1)`index of each integer variable in SOS1 (`INTEGER`). Indices of the i-th SOS1 are stored in `iSOS1(pSOS1(i):pSOS1(i+1))`
*	`rSOS1`	`rSOS1(nSOS1)` reference row of SOS1, storage as for `iSOS1` (`REAL`)
*	`SOS1priority`	`SOS1priority(tSOS1)` priorities of SOS1 sets (`INTEGER`)

	`c`	`c(m)` vector that stores the final values of the general constraints (`REAL`)
*	`cstype`	`cstype(m)` indicates whether the constraint is linear or nonlinear, i.e. `cstype(j) = 'L'` for linear and `cstype(j) = 'N'` for nonlinear constraint number `j` (`CHARACTER*1`)
	`a`	Jacobian storage (see filterSQP) (`REAL`)
	`la`	integer information related to Jacobian storage (see filterSQP) (`INTEGER`)
*	`maxa`	maximum number of entries allowed in Jacobian matrix `a` (`INTEGER`)
	`iwork`	`iwork(liwork)` integer workspace for the MINLP and NLP solvers (`INTEGER`)
*	`liwork`	length of `iwork` (`INTEGER`); at least `nivar + 2*nstackmax + 11` locations *plus* storage required for the NLP solver.
	`work`	`work(lwork)` real workspace for the MINLP and NLP solvers (`REAL`)

*	`lwork`	length of `lwork` (`INTEGER`); at least
		`n+m + nstackmax*(n+m) + nstackmax*n + n`
		`+ 2*nstackmax*nivar + 2*nstackmax + 2*nivar + 3`
		locations *plus* storage required for the NLP solver.

	`iter`	number of NLP problems solved (`INTEGER`)

*	`iprint`	print flag (`INTEGER`) 0 : no printed output; 1 : only result is printed; 2 : result plus intermediary steps are printed; 3 : as 2 but NLP is called with iprint = 1; 4 : as 2 but NLP is called with iprint = 2
*	`nout`	number of output channel (`INTEGER`)
	`ifail`	failure flag (`INTEGER`) 0 : optimal i.f.s. found 1 : infeasible root problem 2 : integer infeasible 3 : stack overflow some i.f.s. obtained 4 : stack overflow, no i.f.s. obtained 5 : SQP termination with `rho < eps` 6 : SQP termination with `iter > max_iter` 7 : crash in user supplied routines 8 : unexpected `ifail` from QP solver 9 : not enough `REAL` workspace or parameter error 10 : not enough `INTEGR` workspace or parameter error
*	`max_NLP`	maximum number of NLP iterations per node (`INTEGER`)

4.2 Common Statements

A number of named common statement are used to pass information into `bqpd` and for less important constants. These common statements take the following form

```
real             eps, infty
common /cTolInf/ eps, infty
```

The common `/cTolInf/` defines the accuracy, `eps`, to which the problem is solved and a suitably large number to represent ∞ in `infty`.

4.3 User-defined Subroutines

The user is also responsible for providing subroutines which compute function, gradient and Hessian information. This is explained in detail in [2].

References

[1] Fletcher, R. and Leyffer, S. Numerical experience with lower bounds for MIQP branch–and–bound. *SIAM Journal on Optimization*, 8(2):604–616, 1998.

[2] Fletcher, R. and Leyffer, S. User manual for filterSQP. Numerical Analysis Report NA/181, Dundee University, April 1998.

[3] H.P. Williams. *Model Solving in Mathematical Programming*. John Wiley & Sons Ltd., Chichester, 1993.